
Computational Optimization and Applications, 30, 147–160, 2005
2005 Springer Science + Business Media, Inc. Manufactured in The Netherlands.

Efficient Algorithms for the Smallest Enclosing Ball Problem

GUANGLU ZHOU* zhouguan@maths.curtin.edu.au
Department of Mathematics and Statistics, Curtin University of Technology, Bentley, WA 6102, Australia

KIM-CHUAN TOH† mattohkc@nus.edu.sg
Department of Mathematics, National University of Singapore, 2 Science Drive 2, Singapore 117543

JIE SUN jsun@nus.edu.sg
Department of Decision Sciences, National University of Singapore, Singapore

Received December 9, 2002; Revised February 26, 2004; Accepted March 10, 2004

Abstract. Consider the problem of computing the smallest enclosing ball of a set of m balls in $\Re^n$. Existing algorithms are known to be inefficient when $n > 30$. In this paper we develop two algorithms that are particularly suitable for problems where n is large. The first algorithm is based on log-exponential aggregation of the maximum function and reduces the problem into an unconstrained convex program. The second algorithm is based on a second-order cone programming formulation, with special structures taken into consideration. Our computational experiments show that both methods are efficient for large problems, with the product mn on the order of 10^7. Using the first algorithm, we are able to solve problems with $n = 100$ and $m = 512,000$ in about 1 hour.

Keywords: computational geometry, smoothing approximation, second order cone programming

AMS Subject Classification: 90C30.

1. Introduction

A ball B_i in $\Re^n$ with center c_i and radius $r_i \geq 0$ is the closed set $B_i = \{x \in \Re^n : \|x - c_i\| \leq r_i\}$. Given a set of balls $\mathcal{B} = \{B_1, B_2, \ldots, B_m\}$ in $\Re^n$, the smallest enclosing ball $mb(\mathcal{B})$ of the set $\mathcal{B}$ is the ball with the smallest radius that encloses all the balls in $\mathcal{B}$. Given a set of points $\mathcal{P} = \{p_1, p_2, \ldots, p_m\}$ in $\Re^n$, the smallest enclosing ball $mp(\mathcal{P})$ of the points is the ball with the smallest radius that encloses all the points in $\mathcal{P}$. Clearly, the problem of smallest enclosing ball of points is a special case of the problem of smallest enclosing ball of balls. In what follows, unless otherwise stated, we will term the problem of smallest enclosing ball of balls to be the smallest enclosing ball problem. This problem can be easily formulated as the following convex optimization problem:

$$\min_{x \in \Re^n} \max_{1 \leq i \leq m} \{\|x - c_i\| + r_i\}. \tag{1}$$

*His work was supported by Australian Research Council.
†Research supported in part by the Singapore-MIT Alliance.

Let

$$f_i(x) = \|x - c_i\| + r_i, \quad i = 1, 2, \ldots, m, \tag{2}$$

and

$$f(x) = \max_{1 \le i \le m} \{f_i(x)\}. \tag{3}$$

Then problem (1) can be rewritten as

$$\min_{x \in \Re^n} f(x). \tag{4}$$

It is readily seen that the solution to (1) exists since $f(x)$ is coercive. Moreover, the solution is unique, for otherwise there would exist two different balls, C_1 and C_2, of the same radius, with $\bigcup_{j=1}^{m} B_j \subset C_i$, $i = 1, 2$. Then one can construct a smaller ball containing $C_1 \cap C_2$ and thus $\mathcal{B}$ as well.

Problem (1) arises in applications such as location analysis and military operations and it is itself of interest as a problem in computational geometry; see [2, 3, 6, 9, 12, 13, 18, 20] for details. Many algorithms have been developed for (1). In particular, for the problem of the smallest enclosing ball of points, Megiddo [9] presented a deterministic $O(m)$ algorithm for the case where $n \le 3$. Welzl [18] developed a simple randomized algorithm with expected linear time in m for the case where n is small, and Gärtner described a C++ implementation thereof in [4]. For the problem of the smallest enclosing ball of balls, we are aware of only a C++ program developed by David White in [19], which is based on Welzl's algorithm [18] and Gärtner's implementation [4]. The existing software packages [4, 19], however, are not efficient for solving problems with $n > 30$. The goal of this paper is to present efficient algorithms that can be used to solve problems where n and m are large (say, on the order of thousands).

Problem (1) is a non-differentiable (nonsmooth) convex optimization problem. Due to the non-differentiability of the objective function, gradient-based algorithms cannot be used to solve this problem. To overcome this difficulty, in Section 2 we give a smooth approximation for the maximum function f. This approximation is based on the so-called log-exponential aggregation function studied, for instance, in [8]. Based on this smooth approximate function, in Section 3 we propose a limited-memory BFGS algorithm for solving the problem. In Section 4, we reformulate (1) as a second order cone programming (SOCP) problem. Thus, we can also apply interior-point methods to solve (1). In Section 5, we report some numerical results, which show the algorithms proposed here are able to solve large problems.

In this paper, unless otherwise stated, all vectors are column vectors. We let I_d denote the $d \times d$ identity matrix. To represent a large matrix with several small matrices, we use semicolons ";" for column concatenation and commas "," for row concatenation. These notations also apply to vectors. We denote the convex hull of a set $S \subseteq \Re^n$ by $\mathrm{co}(S)$. For a convex function $f : \Re^n \to \Re$, we let $\partial f(x)$ denote the subdifferential of f at x. Note that

f is minimized at x over $\Re^n$ if and only if $0 \in \partial f(x)$. For calculus rules on subdifferentials of convex functions, please refer to Chapter VI in [7].

2. Smooth approximation

For any $p > 0$, define the smoothing log-exponential aggregation function of f in (3) as

$$f(x; p) = p \ln \left(\sum_{i=1}^{m} \exp(g_i(x; p)/p) \right) \quad \text{where} \quad g_i(x; p) = r_i + \sqrt{\|x - c_i\|^2 + p^2}. \tag{5}$$

Lemma 1. *The function $f(x; p)$ has the following properties*:

(i) *For any $x \in \Re^n$, and p_1, p_2 satisfying $0 < p_1 < p_2$, we have*

$$f(x; p_1) < f(x; p_2);$$

(ii) *For any $x \in \Re^n$ and $p > 0$, $f(x) \le f(x; p) \le f(x) + p(1 + \ln m)$;*
(iii) *For any $p > 0$, $f(x; p)$ is continuously differentiable and strictly convex.*

Proof: (i) For any $x \in \Re^n$ and p_1, p_2 satisfying $0 < p_1 < p_2$, by Jensen's inequality [5],

$$\begin{aligned} \left[\sum_{i=1}^{m} (\exp(g_i(x; p_2)))^{1/p_2} \right]^{p_2} &> \left[\sum_{i=1}^{m} (\exp(g_i(x; p_2)))^{1/p_1} \right]^{p_1} \\ &> \left[\sum_{i=1}^{m} (\exp(g_i(x; p_1)))^{1/p_1} \right]^{p_1}. \end{aligned}$$

Hence, $f(x; p_1) < f(x; p_2)$.

(ii) Fix $p = p_2$ and let $p_1 \to 0$ in (i), we have $f(x) < f(x; p)$. Let

$$g_\infty(x; p) = \max_{1 \le i \le m} \{g_i(x; p)\}. \tag{6}$$

It is readily proven that $f(x) \le g_\infty(x; p) \le f(x) + p$. Thus, from (5), we have

$$f(x; p) = g_\infty(x; p) + p \ln \sum_{i=1}^{m} \exp\left[(g_i(x; p) - g_\infty(x; p)) / p \right] \le g_\infty(x; p) + p \ln m.$$

Hence

$$f(x) \le f(x; p) \le f(x) + p(1 + \ln m).$$

(iii) For any $p > 0$, clearly, $f(x; p)$ is continuously differentiable. Now we prove that $f(x; p)$ is strictly convex. From (5),

$$\nabla f(x; p) = \sum_{i=1}^{m} \frac{\lambda_i(x; p)}{h_i(x; p)}(x - c_i), \tag{7}$$

where

$$h_i(x; p) = \sqrt{\|x - c_i\|^2 + p^2}, \quad \tau(x; p) = \sum_{i=1}^{m} \exp(g_i(x; p)/p), \tag{8}$$

$$\lambda_i(x; p) = \frac{\exp(g_i(x; p)/p)}{\tau(x; p)}. \tag{9}$$

Let

$$Q_{ij} = \frac{(x - c_i)(x - c_j)^T}{h_i(x; p)h_j(x; p)}, \quad i, j = 1, 2, \ldots, m.$$

From (7), we get

$$\nabla^2 f(x; p) = \sum_{i=1}^{m} \left[\frac{\lambda_i(x; p)}{h_i(x; p)}(I_n - Q_{ii}) + \frac{\lambda_i(x; p)}{p} Q_{ii} - \sum_{j=1}^{m} \frac{\lambda_i(x; p)\lambda_j(x; p)}{p} Q_{ij} \right].$$

For any $z \in \Re^n$ with $z \neq 0$, by the Cauchy-Schwartz inequality,

$$\|z\|^2 - z^T Q_{ii} z \geq \|z\|^2 - \|z\|^2 \|(x - c_i)/h_i(x; p)\|^2 > 0, \quad \forall\, i = 1, \ldots, m.$$

Thus,

$$\begin{aligned} z^T \nabla^2 f(x; p) z &= \sum_{i=1}^{m} \frac{\lambda_i(x; p)}{h_i(x; p)}(\|z\|^2 - z^T Q_{ii} z) \\ &\quad + \sum_{i=1}^{m} \left[\frac{\lambda_i(x; p)}{p} z^T Q_{ii} z - \sum_{j=1}^{m} \frac{\lambda_i(x; p)\lambda_j(x; p)}{p} z^T Q_{ij} z \right] \\ &> \sum_{i=1}^{m} \left[\frac{\lambda_i(x; p)}{p} z^T Q_{ii} z - \sum_{j=1}^{m} \frac{\lambda_i(x; p)\lambda_j(x; p)}{p} z^T Q_{ij} z \right] \\ &= \frac{1}{p} \sum_{i=1}^{m} \lambda_i(x; p) a_i^2 - \frac{1}{p} \left(\sum_{i=1}^{m} \lambda_i(x; p) a_i \right)^2 \\ &\geq 0, \end{aligned} \tag{10}$$

where $a_i = z^T(x - c_i)/h_i(x; p)$. This shows that $\nabla^2 f(x; p)$ is positive definite. Therefore, $f(x; p)$ is strictly convex. Note that the inequality (10) follows from the fact that

$$\left|\sum_{i=1}^{m} \lambda_i(x; p)a_i\right| \leq \sqrt{\sum_{i=1}^{m} \lambda_i(x; p)} \sqrt{\sum_{i=1}^{m} \lambda_i(x; p)a_i^2},$$

and $\sum_{i=1}^{m} \lambda_i(x; p) = 1$, $\lambda_i(x; p) \geq 0$ for $i = 1, \ldots, m$. □

3. Log-exponential aggregation algorithm

We now give an algorithm for problem (1) based on Lemma 1, followed by a global convergence result.

Algorithm 1.

Let $\sigma \in (0, 1)$, $x^0 \in \Re^n$ and $p^0 > 0$ be given, and set $k := 0$.
For $k = 0, 1, 2, \ldots,$ **do**

1. Use an unconstrained minimization method to solve

$$\min_{x \in \Re^n} f(x; p^k). \tag{11}$$

   Let the minimizer be x^k.
2. Set $p^{k+1} = \sigma p^k$, increment k by 1, and return to Step 1.

End

Lemma 2. *Let $\{x^k\}_{k\geq 1}$ be the sequence of points produced by Algorithm* 1. *Then any limit point of $\{x^k\}_{k\geq 1}$ is an optimal solution of* (1).

Proof: Let x^* be a limit point of $\{x^k\}_{k\geq 1}$. Without loss of generality, we suppose that $x^k \to x^*$ as k tends to $+\infty$. From the fact that x^k is a solution of (11), we have

$$\nabla f(x^k; p^k) = \sum_{i=1}^{m} \frac{\lambda_i^k}{h_i(x^k; p^k)}(x^k - c_i) = 0,$$

where $\lambda_i^k = \lambda_i(x^k; p^k)$, and $h_i(x; p^k)$ is defined as in (8). Let

$$I(x^*) = \{i : f_i(x^*) = f(x^*), i = 1, 2, \ldots, m\}.$$

By noting that

$$\lambda_i^k = \frac{\exp[(g_i(x^k; p^k) - g_\infty(x^k; p^k))/p^k]}{\sum_{i=1}^{m} \exp[(g_i(x^k; p^k) - g_\infty(x^k; p^k))/p^k]},$$

we have

$$\lim_{k\to+\infty} \lambda_i^k = 0, \quad i \notin I(x^*). \tag{12}$$

By (12) and the fact that $\sum_{i=1}^m \lambda_i^k = 1$ and $\lambda_i^k > 0$, $i = 1, 2, \ldots, m$, the sequence $\{\lambda_i^k, i = 1, 2, \ldots, m\}_{k\geq 1}$ has a convergent subsequence. Without loss of generality, we suppose that

$$\lim_{k\to+\infty} \lambda_i^k = \lambda_i^*, \quad i \in I(x^*).$$

Then $\sum_{i\in I(x^*)} \lambda_i^* = 1$ and $\lambda_i^* \geq 0$, $i \in I(x^*)$. For $i \in I(x^*)$,

$$t_i^* = \lim_{k\to+\infty} \frac{x^k - c_i}{h_i(x^k; p^k)} \in \partial f_i(x^*).$$

Thus

$$\lim_{k\to+\infty} \nabla f(x^k; p^k) = \sum_{i\in I(x^*)} \lambda_i^* t_i^* = 0.$$

This shows that $0 \in \mathrm{co}(\partial f_i(x^*), i \in I(x^*)) = \partial f(x^*)$. Therefore, x^* is an optimal solution of (1). □

Theorem 3. *Let $\{x^k\}_{k\geq 1}$ be the sequence of points produced by Algorithm* 1 *and x^* be the unique optimal solution of* (**1**)*. Then*

$$\lim_{k\to+\infty} x^k = x^*.$$

Proof: For any $k \geq 1$, by Lemma 1,

$$f(x^1; p^1) > f(x^1; p^k) \geq f(x^k; p^k) \geq f(x^k). \tag{13}$$

Since $f(x)$ is coercive, the level set

$$L = \{x \in \Re^n : f(x) \leq f(x^1; p^1)\}$$

is bounded. From (13), we have $\{x^k\}_{k\geq 1} \subseteq L$. Hence $\{x^k\}_{k\geq 1}$ is bounded. As x^* is the unique solution of (1), it follows from Lemma 2 that

$$\lim_{k\to+\infty} x^k = x^*.$$

□

In practice, we would stop Algorithm 1 once some stopping criteria are met. Moreover, we will use a first-order, or gradient based, unconstrained minimization algorithm for solving

(11) in Step 1. In the following algorithm, we will choose a limited-memory BFGS algorithm [1] to solve (11) as this algorithm can solve very large unconstrained optimization problems efficiently. The following is a more practical version of Algorithm 1.

Algorithm 2.

Let $\sigma \in (0, 1)$, $\epsilon_1, \epsilon_2 > 0$, $x^0 \in \Re^n$ and $p^0 > 0$ be given, and set $k := 0$.
For $k = 0, 1, 2, \ldots,$ until $p^k \leq \epsilon_1$, **do**

1. Use a version of the limited-memory BFGS algorithm to solve (11) approximately, and obtain an x^k such that $\|\nabla f(x^k; p^k)\| \leq \epsilon_2$.
2. Set $p^{k+1} = \sigma p^k$, increment k by 1, and return to Step 1.

End
The actual parameter values used for $p^0, \sigma, \epsilon_1, \epsilon_2$ are given in Section 5.

4. SOCP reformulation

Problem (1) is equivalent to the following problem.

$$\begin{array}{ll} \min & r \\ \text{s.t.} & \|y - c_i\| + r_i \leq r, \quad i = 1, \ldots, m, \\ & y \in \Re^n. \end{array} \tag{14}$$

It turns out that this can be reformulated as follows:

$$\begin{array}{ll} \max & -r \\ \text{s.t.} & -t_i + r = r_i, \\ & -s_i + y = c_i, \\ & t_i \geq \|s_i\|, \quad i = 1, \ldots, m. \end{array} \tag{15}$$

Let

$$\begin{aligned} &N = m(n+1), \quad K := \{(t; s) \in R^{n+1} : \ t \geq \|s\|\}, \quad \boldsymbol{y} = (r; y) \in R^{n+1}, \\ &\boldsymbol{c} = -(r_1; c_1; r_2; c_2; \ldots; r_m; c_m) \in R^N, \quad \boldsymbol{A} = -(I_{n+1}, \ldots, I_{n+1}) \in R^{(n+1)\times N}, \\ &\boldsymbol{b} = -(1; 0; \ldots; 0) \in R^{n+1}, \quad \boldsymbol{s} = (t_1; s_1; t_2; s_2; \ldots; t_m; s_m) \in R^N, \end{aligned}$$

where I_{n+1} is an identity matrix. For later purpose, we use the notation $\gamma(\alpha; \beta) = \sqrt{\alpha^2 - \|\beta\|^2}$ for $(\alpha; \beta) \in K$. The problem (15) can be written as a standard dual second order cone program as follows:

$$\begin{array}{lll} (D) & \max & \boldsymbol{b}^T \boldsymbol{y}, \\ & \text{s.t.} & \boldsymbol{s} = \boldsymbol{c} - \boldsymbol{A}^T \boldsymbol{y}, \\ & & \boldsymbol{s} \in \mathcal{K}, \end{array}$$

where $\mathcal{K} = K \times K \times \cdots \times K = K^m$. Let

$$\boldsymbol{x} = (\tau_1; x_1; \tau_2; x_2; \ldots; \tau_m; x_m) \in R^N.$$

The corresponding primal problem is

$$\begin{array}{lll} (P) & \min & \boldsymbol{c}^T\boldsymbol{x} \\ & \text{s.t.} & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \\ & & \boldsymbol{x} \in \mathcal{K}. \end{array}$$

This pair of problems (P) and (D) are studied in Nesterov and Nemirovskii [10], Nesterov and Todd [11], and Xue and Ye [21]. We can solve these problems by using a primal-dual interior-point method (IPM).

There are IPM based general purpose softwares for solving SOCPs, examples are SeDuMi [14], SDPT3 [16], and LOQO [17]. But the problems (P) and (D) have a very special constraint matrix $\boldsymbol{A}$, and it is beneficial to analyze how such a structure can be exploited when computing the search direction in each IPM iteration. Suppose that at a particular IPM iteration, the current iterate is $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{s})$ and the target barrier parameter is μ. To obtain the next iterate, the most expensive computation lies in assembling and solving a linear system of $n+1$ equations for the search direction Δy. By adapting the results from Section 2.3 of [16] for the so-called Nesterov-Todd search direction, such a linear system takes the following form for our present SOCPs:

$$\underbrace{\left(\frac{1}{\sigma^2}J + \sum_{i=1}^{m}\frac{1}{w_i^2}\boldsymbol{g}_i\boldsymbol{g}_i^T\right)}_{M}\ \Delta\boldsymbol{y} \ = \ h, \tag{16}$$

where

$$J = \begin{bmatrix} -1 & \\ & I_n \end{bmatrix}, \quad w_i^2 = \frac{\gamma(t_i; s_i)}{\gamma(\tau_i; x_i)}, \quad \frac{1}{\sigma^2} = \sum_{i=1}^{m}\frac{1}{w_i^2},$$

$$\boldsymbol{g}_i = \frac{\sqrt{2}}{\gamma(\rho_i; \xi_i)}(-\rho_i; \xi_i), \quad (\rho_i; \xi_i) \ = \ \frac{1}{w_i}(t_i; s_i) - w_i(-\tau_i; x_i),$$

and

$$h = \boldsymbol{b} + \sum_{i=1}^{m}\frac{1}{w_i^2}\left(J + \boldsymbol{g}_i\boldsymbol{g}_i^T\right)((r_i; c_i) + (t_i; s_i) - \boldsymbol{y}) - \sum_{i=1}^{m}\frac{\mu}{\gamma(t_i; s_i)^2}(-t_i; s_i).$$

Note that the matrix M is symmetric positive definite. In the above, we give a full description of the linear system (16) for the sake of completeness. But our main aim is to study the

matrix M, and explore the possibility of solving (16) via a Krylov subspace method such as the preconditioned conjugate gradient (PCG) method when n is large.

Let

$$\mathcal{I} \;=\; \left\{ i \in \{1,\ldots,m\} : \frac{\sigma \|\boldsymbol{g}_i\|}{w_i} > 1 \right\}, \quad \mathcal{J} \;=\; \{1,\ldots,m\} \setminus \mathcal{I}.$$

We can write the matrix M in (16) as follows:

$$M \;=\; \underbrace{\frac{1}{\sigma^2}\left(I \;+\; \sum_{i\in\mathcal{I}} \frac{\sigma^2}{w_i^2} \boldsymbol{g}_i \boldsymbol{g}_i^T \right)}_{M_0} + \underbrace{\frac{1}{\sigma^2} \sum_{j\in\mathcal{J}} \frac{\sigma^2}{w_j^2} \boldsymbol{g}_j \boldsymbol{g}_j^T}_{M_1} \;-\; \frac{2}{\sigma^2} e_1 e_1^T,$$

where M_0 is positive definite, M_1 is positive semidefinite, and $e_1 = (1;0;\ldots;0) \in \Re^{n+1}$. Numerical experiments have shown that the cardinality of $\mathcal{I}$ is usually much smaller than n when the barrier parameter μ is small. Thus M_0 is a diagonal matrix plus a low-rank matrix, and its inverse can be found readily via the Sherman-Morrison-Woodbury formula:

$$M_0^{-1} = \sigma^2 \underbrace{[I - G(I + G^T G)^{-1} G^T]}_{B}, \quad G = \left(\frac{\sigma}{w_i} \boldsymbol{g}_i \;\middle|\; i \in \mathcal{I} \right).$$

Suppose M_0 is used as a preconditioner in (16), then the preconditioned matrix $M_0^{-1} M$ has the form:

$$M_0^{-1} M \;=\; I + \frac{1}{\sigma^2} M_0^{-1} M_1 - \frac{2}{\sigma^2} M_0^{-1} e_1 e_1^T \;=\; I + B M_1 - 2 B e_1 e_1^T. \tag{17}$$

To analyze the spectrum of $M_0^{-1} M_1$, we make use of the following lemma.

Lemma 4. *Let $p = |\mathcal{I}|$. If $p < n+1$, then the matrix B is positive semidefinite and*

$$\|B\| \leq 1$$

Proof: Let the singular value decomposition of $G \in \Re^{(n+1)\times p}$ be

$$G \;=\; U \Sigma V^T,$$

where $U \in \Re^{(n+1)\times p}$ has orthonormal columns, $\Sigma \in R^{p\times p}$ is diagonal, and $V \in \Re^{p\times p}$ is orthogonal. It is easily shown that

$$G(I + G^T G) G^T \;=\; U \Sigma (I + \Sigma^2)^{-1} \Sigma U^T.$$

Thus

$$B = I - U\Sigma(I + \Sigma^2)^{-1}\Sigma U^T$$
$$= (I - UU^T) + U(I + \Sigma^2)^{-1}U^T, \tag{18}$$

which is the sum of two positive semidefinite matrices, and hence B is also positive semidefinite. From (18), it is clear that $x^T Bx \leq x^T x$ for any $x \in \Re^{n+1}$. Hence, $\|B\| \leq 1$. □

Notice that BM_1 is similar to a positive semidefinite matrix and thus all its eigenvalues lie in the interval $[0, \|BM_1\|] \subset [0, \|M_1\|]$. For $j \in \mathcal{J}$, $\sigma\|\boldsymbol{g}_j\|/w_j \leq 1$, thus $\|M_1\| \leq |\mathcal{J}|$, and we would expect $\|M_1\|$ to be a moderate number. Hence, we can expect the preconditioned matrix $M_0^{-1}M$ to have a favorable spectral distribution (with all its eigenvalues clustered around 1 except possibly for an outlier due to the presence of a rank-1 perturbation in (17)) for the PCG to attain fast convergence.

So far, we have discussed the preconditioning strategy to solve (16) when the barrier parameter μ is small. When μ is not small, the cardinality of $\mathcal{I}$ may be comparable to n, and it would be computationally expensive to use M_0 as a preconditioner. Fortunately when μ is not small, the matrix M itself is quite well-conditioned and preconditioning by the diagonal of M would be good enough for PCG to obtain fast convergence. In practice, we switch from the diagonal preconditioner to M_0 when

$$\frac{\max(\|\boldsymbol{g}_i\|/w_i)}{\min(\|\boldsymbol{g}_i\|/w_i)} \;\geq\; 5 \times 10^2. \tag{19}$$

5. Computational results

We implemented Algorithm 2 described in Section 3 and the numerical experiments were done by using a Pentium IV 2.2 GHz personal computer with 4 GB of memory. To solve (11), we choose a limited-memory BFGS algorithm with strong Wolfe-Powell line-search and 7 limited-memory vector updates [1]. We implemented our code in Fortran 77 and compared it with SeDuMi (version 1.05) [14] and SDPT3 (version 3.02) [16] (two high quality software packages with MATLAB interface for solving semidefinite-quadratic-linear programs). We also implemented an adaptation of SDPT3 to exploit the special structure present in our SOCPs in (P) and (D). The main change lies in replacing the direct solution method in computing the direction Δy in SDPT3 by the PCG iterative solution method discussed in the last section. For convenience, we will call the modified algorithm SDPT3-PCG.

Throughout the computational experiments, we use the following parameters in Algorithm 2:

$$p^0 = 0.01, \quad \sigma = 0.1, \quad \epsilon_1 = 10^{-6}, \quad \text{and} \quad \epsilon_2 = 10^{-5}.$$

The starting point is $x^0 = \mathbf{0}$. For SDPT3, we stop the interior-point algorithm when ϕ is smaller than 10^{-6}. Here

$$\phi \;=\; \max\left(\frac{\|\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}\|}{1 + \|\boldsymbol{b}\|},\ \frac{\|\boldsymbol{c} - \boldsymbol{s} - \boldsymbol{A}^T\boldsymbol{y}\|}{1 + \|\boldsymbol{c}\|},\ \frac{\boldsymbol{x}^T\boldsymbol{s}}{1 + 0.5(|\boldsymbol{b}^T\boldsymbol{y}| + |\boldsymbol{c}^T\boldsymbol{x}|)}\right).$$

We use the same stopping criterion for SDPT3-PCG. When solving the system (16), we stop the PCG iteration when the relative residual norm, $\|h - M\Delta\mathbf{y}\|/\|h\|$, is less than 10^{-8}. For SeDuMi, we use all the default values except that the parameter `eps` is set to be 10^{-6}.

The test problems are generated randomly. We use the following pseudo-random sequence:

$$\psi_0 = 7, \quad \psi_{i+1} = (445\psi_i + 1) \bmod 4096, \quad i = 1, 2, \ldots,$$
$$\bar{\psi}_i = \frac{\psi_i}{40.96}, \quad i = 1, 2, \ldots. \tag{20}$$

The elements of c_i, $i = 1, 2, \ldots, m$, are successively set to $\bar{\psi}_1, \bar{\psi}_2, \ldots$, in the order:

$$r_1,\ c_1(1),\ c_1(2), \ldots,\ c_1(n),\ r_2,\ c_2(1), \ldots,\ c_2(n), \ldots, r_m,\ c_m(1), \ldots,\ c_m(n).$$

The numerical results we obtained are summarized in Tables 1–4. In these tables, **n** and **m** denote the dimension of the Euclidean space and the number of balls, respectively, **Obj Value** denotes the value of the objective function at the final iteration, **Iter** denotes the number of iterations, **Time** denotes the CPU time in second for solving each problem, and **RAM** denotes the random access memory usage in MB.

Table 1. Objective function value at the final iteration.

Problem		Obj value			
n	*m*	SeDuMi	SDPT3	Algorithm 2	SDPT3-PCG
400	1000	6.79603198e2	6.79603173e2	6.79603176e2	6.79603173e2
800	1000	9.16972254e2	9.16972204e2	9.16972207e2	9.16972204e2
1200	1000	1.10067761e3	1.10067759e3	1.10067760e3	1.10067759e3
1600	1000	1.25331993e3	1.25331987e3	1.25331987e3	1.25331987e3
2000	1000	1.39062921e3	1.39062927e3	1.39062919e3	1.39062927e3

Table 2. Performance comparison of SeDuMi, SDPT3, Algorithm 2 and SDPT3-PCG. The numbers in the last two columns correspond to average value of $|\mathcal{I}|$ and the average number of PCG iterations in each interior-point iteration, respectively.

Problem		SeDuMi		SDPT3		Algorithm 2		SDPT3-PCG			
n	*m*	Time	Iter	Time	Iter	Time	Iter	Time	Iter		
400	1000	83.2	15	69.7	17	33.3	795	12.1	17	78	12
800	1000	256.0	15	194.8	16	69.6	848	22.1	16	91	16
1200	1000	499.4	15	274.6	15	77.9	572	30.8	15	94	17
1600	1000	1420.2	15	374.7	15	118.7	716	41.4	15	110	15
2000	1000	1433.0	15	417.7	13	144.9	638	45.2	13	120	17

Table 3. Performance of Algorithm 2 and SDPT3-PCG on problems with large n.

Problem		Algorithm 2			SDPT3-PCG		
n	m	Time	Iter	RAM	Time	Iter	RAM
1000	3000	246.7	842	23	84.3	16	403
2000	3000	514.5	883	46	170.3	15	863
4000	3000	1012.3	849	92	400.4	15	1680
8000	3000	2127.7	889	185	Out of memory		
16000	3000	4168.8	841	369	Out of memory		

Table 4. Performance of Algorithm 2 and SDPT3-PCG on problems with large m.

Problem		Algorithm 2			SDPT3-PCG		
n	m	Time	Iter	RAM	Time	Iter	RAM
100	8000	60.9	748	7	25.2	18	123
100	16000	116.0	716	13	56.2	21	253
100	32000	233.8	713	26	115.9	21	478
100	64000	490.8	753	51	260.5	22	942
100	128000	965.5	732	100	592.7	23	1726
100	256000	1968.2	742	199	Out of memory		
100	512000	3765.7	715	398	Out of memory		

The results reported in Tables 1 and 2 show that algorithms presented in this paper perform very well. We can see from Table 1 that all algorithms are able to obtain good accuracy for moderately large problems. Moreover, it appears from Table 2 that Algorithm 2 consistently takes more iterations than the SOCP algorithm and that for all tested problems, Algorithm 2 and SDPT3-PCG consistently use less CPU time than SeDuMi and SDPT3. It is interesting to note that for the last 2 problems, SeDuMi, SDPT3, and Algorithm 2 are about 30, 8.5, and 2.8 times more expensive (in CPU time) than SDPT3-PCG, respectively. From the performance data in Table 2, it is clear that for the smallest enclosing ball problem, Algorithm 2 and SDPT3-PCG are much more efficient than the general purpose algorithms in SeDuMi or SDPT3. Thus we shall concentrate only on Algorithm 2 and SDPT3-PCG for the rest of this section.

Next, we look at the performance of Algorithm 2 and SDPT3-PCG for problems with large n but m is moderate. For each pair of (n, m), we run the algorithms on 5 random problems generated from (20) with the integer 445 replaced by 437, 441, 445, 449, and 453. The results (each number is the average over 5 problems) reported in Table 3 show that Algorithm 2 can solve problems with very large n in a reasonable amount of CPU time that grows linearly with n. Its memory demand is just slightly more than the amount required to store the data points $\{c_i \in \Re^n : i = 1, \ldots, m\}$. However, SDPT3-PCG fails for the last

2 larger problems in Table 3 due to excessive memory requirement. In fact, SDPT3-PCG requires about 18 times more memory space than Algorithm 2. It is interesting to note however, that when there is enough computer memory, SDPT3-PCG appears to be more efficient (in terms of CPU time) than Algorithm 2.

In Table 4, we compare the the performance of Algorithm 2 and SDPT3-PCG for problems with large m but n is moderate. Again, we test the algorithms on 5 randomly generated problems for each pair of (n, m). Both Algorithm 2 and SDPT3-PCG can solve these problems very efficiently. The required CPU time and memory for Algorithm 2 grow almost linearly with m. For SDPT3-PCG, the required CPU time grows like $m^{1.13}$ with m. Again, its memory demand is about 18 times more than that required by Algorithm 2.

From the results in Tables 2 to 4, we conclude that Algorithm 2 is better than SeDuMi, SDPT3 and SDPT3-PCG for very large problems, since it consumes less memory. For moderately large problems, SDPT3-PCG is more efficient (in CPU time) than Algorithm 2.

References

1. R.H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," SIAM J. Sci. Comput., vol. 16, pp. 1190–1208, 1995.
2. J. Elzinga and D. Hearn, "The minimum covering sphere problem," Management Sci., vol. 19, pp. 96–104, 1972.
3. B. Gärter, "Fast and robust smallest enclosing balls," in Algorithms-ESA'99: 7th Annual European Symposium Proceedings, J. Nestril (Ed.), Lecture Notes in Computer Science 1643, Springer-Verlag, pp. 325–338.
4. B. Gärter, Smallest enclosing ball—Fast and robust in C++, http://www.inf.ethz.ch/personal/gaertner/miniball.html.
5. G.H. Hardy, J.E. Littlewood, and G. Polya, Inequalities. Cambridge University Press, 1952.
6. D.W. Hearn and J. Vijan, "Efficient algorithms for the minimum circle problem," Oper. Res., vol. 30, pp. 777–795, 1982.
7. J.B. Hiriart-Urruty and C. Lemarechal, Convex Analysis and Minimization Algorithm. Springer-Verlag: Berlin, Heidelberg, 1993.
8. X.S. Li, "An aggregate function method for nonlinear programming," Sci. China Ser. A, vol. 34, pp. 1467–1473, 1991.
9. N. Megiddo, "Linear-time algorithms for linear programming in $\Re^3$ and related problems," SIAM J. Comput., vol. 12, pp. 759–776, 1983.
10. Yu. E. Nesterov and A. Nemirovskii, Interior Polynomial Algorithms in Convex Programming. SIAM: Philadelphia, 1994.
11. Yu. E. Nesterov and M.J. Todd, "Self-scaled barriers and interior-point methods for convex programming," Math. Oper. Res., vol. 22, pp. 1–42, 1997.
12. F.P. Preparata and M.I. Shamos, Computational Geometry: An Introduction, Texts and Monographs in Computer Science, Springer-Verlag: New York, 1985.
13. M.I. Shamos and D. Hoey, "Closest-point problems," in 16th Annual Symposium on Foundations of Computer Science (Berkeley, CA, 1975), IEEE Computer Society, Long Beach, CA, 1975, pp. 151–162.
14. J.F. Sturm, "Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones," Optim. Methods Softw., vol. 11 & 12, pp. 625–653, 1999.
15. P. Sun and R.M. Freund, "Computation of minimum volume covering ellipsoids," MIT Operations Research Center Working Paper OR 064-02, July, 2002.
16. R.H Tütüncü, K.C. Toh, and M.J. Todd, "Solving semidefinite-quadratic-linear programs using SDPT3," Math. Programming, vol. 95, pp. 189–217, 2003.
17. R.J. Vanderbei, "LOQO user's manual—version 3.10," Optim. Methods Softw., vol. 11 & 12, pp. 485–514, 1999.

18. E. Welzl, "Smallest enclosing disks (balls and ellipsoids)," in New Results and New Trends in Computer Science, H. Maurer (Ed.) Springer-Verlag, 1991, pp. 359–370.
19. D. White, Enclosing ball software, http://vision.ucsd.edu/˜dwhite/ball.html.
20. S. Xu, R. Freund, and J. Sun, "Solution methodologies for the smallest enclosing circle problem, "Comput. Optim. Appl.," vol. 25, pp. 283–292, 2003.
21. G.L. Xue and Y.Y. Ye, "An efficient algorithm for minimizing a sum of Euclidean norms with applications," SIAM J. Optim., vol. 7, pp. 1017–1036, 1997.